UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO DE INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE SERVICIOS INTERACTIVOS A TRAVÉS DE TELÉFONOS INTELIGENTES PARA UN TABLÓN EXPOSITOR CON ILUMINACIÓN LED

JAVIER POMEDA MONROY

2016

TRABAJO FIN DE GRADO

TÍTULO: **Desarrollo de servicios interactivos a través de teléfonos inteligentes para un tablón expositor con iluminación LED.**

AUTOR: **D. Javier Pomeda Monroy.**

TUTORA: **Dña. Elena Romero Perales.**

PONENTE: **D. Octavio Nieto-Taladriz García**

DEPARTAMENTO: **Departamento de Ingeniería Electrónica.**

TRIBUNAL:

Presidente: **D. Carlos López Barrio.**

Vocal: **D. Rubén San Segundo Hernández.**

Secretario: **D. Alvaro Araujo Pinto.**

Suplente: **D. Giorgios Kontaxakis.**

FECHA DE LECTURA: _____

CALIFICACIÓN: _____

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO DE INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Desarrollo de servicios interactivos a través de teléfonos inteligentes para un tablón expositor con iluminación LED.

Javier Pomeda Monroy

2016

Agradecimientos

Llega el momento de terminar la carrera, y con él, se cierra una etapa de mi vida. Pero esto no habría sido posible sin la ayuda y el apoyo de muchas personas a las que quiero agradecerles:

En primer lugar, a mis padres, por ser los cimientos sobre los que construiré el resto de mi vida. Especialmente a mi madre, por su incondicional cariño y dedicación desde el minuto uno y hasta hoy. En general, también al resto de mi familia por los mismos motivos.

Después, a Ale, por ser un apoyo incondicional siempre, siempre, siempre, y por todos los buenos momentos que nos quedan por llegar.

A todos mis amigos, incluyendo a los zoleros: Andrea, Chol, Esteban, Imp, que ya son años contando anécdotas que nunca nos cansarán. Y a la puta Jet Set, que no me olvido de Álvaro. Los músicos de la gran y mítica Guitar Overdose (grande José Luis) y relacionados incluidos en el grupo Seta: Laura, Yorch, Zaida, Kike, Rodri, Casas, Pablo; a todos mis colegis de la universidad, los de casi siempre: Rafa, Dani, Ángel, y toda la inmensa gente del Eco de Teleco que lo cambiaron todo: Borja, Carlos, David, Diego, Héctor, Zhan, Edu, Pulpo, Helio, Gis, Isa, Grande Javi, José, Manu, Moi, Nacho, Noe, Quique, Rober, Saíd, Sonia, Vir, JaviBro, Jaime, Gon, Imad, Loren y Yaiza. Irrepetibles desde el primer redactor al último. Y sí, lo he copiado del whatsapp, antes de que salte algún listo.

A Microsoft Word por absolutamente nada.

A mis compañeros Erasmus, que puede que nunca lean esto, pero que compartieron la mejor experiencia que tuve jamás y merecen un hueco: Anna, Louise, Héctor, Bea, Laura, Luca y Ádam.

A todos mis compañeros de trabajo en National Instruments por acogerme durante estos últimos meses y la oportunidad de trabajar con tan grandes personas.

A Flipout por la promesa de un paraíso terrenal.

En general, a los haters por pasar de estado líquido a gaseoso habitualmente.

A toda la gente del B105, incluyendo rookies y mayores, por permitirme seguir con 100% de efectividad en conocer buena gente en los sitios a los que voy. A mi tutora, Elena, por supuesto, que siempre me ayudó a pesar de mis continuas y desastrosas planificaciones. Por supuesto, también a la Pericracia por su transparencia y honorabilidad en el torneo de futbolín.

Y por último, a un viejo amigo que dejó teleco antes de tiempo, en el cual cobran más sentido todas estas palabras. Prometí que llevaría la carga y acabaría por los dos. Es Bolonia, tendrás que conformarte, pero ahí lo llevas. No me olvido: tenemos pendientes un par de partidos y un último concierto cuando nos veamos.

No se trata de ninguna fruta del Diablo, ni de ningún truco fácil. Uno tras otro, convierte a todos con los que le rodean en sus aliados. Más que nadie que haya surcado nunca los mares, ese hombre posee el más grande de todos los poderes.

Resumen

El tablón expositor del laboratorio B-105 cuenta con un sistema de iluminación mediante tiras de LED de tres canales RGB, controlados mediante un circuito integrado de Texas Instruments, el TLC5940.

El objetivo del Trabajo de Fin de Grado consistía en el desarrollo de funcionalidades que permitan poder controlar los efectos de luces del tablón mediante un teléfono inteligente con sistema operativo Android. Para ello se ha hecho uso de conexión Bluetooth entre el teléfono y el sistema empotrado que controla los leds, una Raspberry Pi.

Se han realizado las siguientes fases del proyecto:

- Diseño e implementación del código a ejecutar en un sistema Raspberry Pi en C++, con el objeto de crear diferentes efectos de los diodos LED y funcionalidades para el tablón:
    - Codificación de los diversos efectos de luces.
    - Diseño del código e interrupciones para la conexión Bluetooth, utilizando laslibrerías disponibles para Raspberry Pi.
- Diseño e implementación de la aplicación para teléfonos Android:
    - Diseño e implementación de la interfaz de usuario, incluyendo la posibilidad de personalización para los diferentes efectos.
    - Implementación de la conectividad Bluetooth desde la aplicación utilizando las librerías proporcionadas por Android.
    - Desarrollo del código Java que gestiona la lógica de la aplicación.
- Pruebas y validación del proyecto.

Las diferentes fases se han enfocado con vistas a la posibilidad de ampliar el proyecto en el futuro añadiendo nuevas alternativas de control y uso tanto a la Raspberry Pi como a la aplicación móvil.

Summary

The B-105 Electronic Systems Lab pinboard has around its framework an illumination system which is made up of LED strips of three RGB channels controlled by a Texas Instruments driver, the TLC5940.

The goal of the thesis was at first the development of functionalities which would allow the modification and personalization of the light effects utilizing an Android smartphone. To achieve this objective, a Bluetooth connection was implemented between the smartphone and the embedded system that manages the LED stripes, a Raspberry Pi.

We have developed the different phases of the project as:

- Design and implementation of the C++ code which will be executed on the Raspberry Pi, with the goal of personalizing different light effects in the LED diodes and functionalities for the pinboard:
    - Codification of the different light effects.
    - Code design and interruptions for the Bluetooth connection, using the available libraries for Raspberry Pi.
- Design and implementation of the Android app:
    - Design and implementation of the user interface, including the possibility of personalization for the different effects.
    - Implementation of the Bluetooth connectivity from the app using the Android available libraries.
    - Development of the Java code which manages the application logic.
- Tests and validation of the project.

The different phases were focused to the possibility of extending it in the future, adding new control and new utilization alternatives both for the Raspberry Pi and the Android app.

Palabras clave

Android, Raspberry Pi, sistemas empotrados, Bluetooth, C++, interfaz de usuario, Java, PCB, LED, circuitos integrados.

Keywords

Android, Raspberry Pi, embedded, Bluetooth, C++, user interface, Java, PCB, LED, integrated circuits.

Índice

1. Introducción.

1.1. Preámbulo.

El laboratorio de investigación del B-105 Electronic Systems Lab, cuenta en su exterior con un tablón expositor de las diferentes actividades realizadas en el mismo, así como ejemplos de sus trabajos realizados.

Ilustración 1. Tablón expositor.

El tablón cuenta con un sistema de iluminación mediante dieciséis tiras de LED de tres canales RGB. Todo el diseño está controlado por un sistema embebido, en el cual se configuran diferentes efectos de luces para el tablón.

El objetivo del tablón es atraer la atención de las personas que pasen por el pasillo y puedan interesarse por la actividad del laboratorio. Las luces LED, además de dotar de una cierta estética al tablón, representan la actividad principal del laboratorio: la electrónica.

1.2. Objetivos.

El objetivo de este Trabajo Fin de Grado consistirá en el desarrollo de funcionalidades interactivas, mediante la especificación Bluetooth, que permita el control de los efectos

de luces del tablón con un teléfono inteligente con sistema operativo Android. Así, no solo conseguiremos un control de la iluminación del tablón a distancia, sino también la interacción del usuario, pudiendo personalizar de este modo las luces a nuestro criterio.

Ilustración 2. Esquema del proyecto.

Para ello se seguirán los siguientes pasos:

1. Análisis del sistema previo.
2. Familiarización y tutoriales con Raspberry Pi.
3. Diseño del nuevo sistema y elección del módulo Bluetooth.
4. Codificación de distintos efectos de luces del tablón.
5. Diseño de una interfaz de usuario de Android.
6. Implementación de la comunicación Bluetooth en Android.
7. Diseño e implementación de la comunicación Bluetooth Raspberry - Android.
8. Realización de pruebas y validación del proyecto.

A lo largo del desarrollo del proyecto se tocarán otros temas relacionados de forma directa o indirecta con el mismo, tales como conceptos de redes, programación en C++, fundamentos de sistemas Linux, diseño de sistemas electrónicos, soldadura y montaje de componentes de inserción y SMD, protocolos de comunicaciones, etc.

1.3. Estructura de la memoria.

A lo largo de la memoria se expondrán las diferentes fases de las que consta el proyecto, partiendo de una visión general y continuando hacia los detalles y particularidades de cada uno de los elementos que lo componen. En el capítulo 2 abordaremos el análisis del sistema previo, incluyendo tanto el hardware que se encontraba inicialmente en el mismo además del software en el que se ha basado el desarrollo de nuevas

funcionalidades, mientras que en el capítulo 3 se dará a conocer el nuevo sistema y los cambios realizados a nivel hardware en el mismo.

La parte relativa al diseño software en Raspberry Pi se comentará en el capítulo 4. Primero encontraremos el desarrollo de nuevas funcionalidades, que incluyen el código final a ejecutar en el programa principal; a continuación se explicará detalladamente todo lo que concierna a la comunicación Bluetooth.

Abordaremos todo lo relativo a la aplicación Android, tanto la interfaz de usuario como la lógica de la aplicación, así como lo relativo a la comunicación Bluetooth desde el terminal móvil, dentro del capítulo 6.

A continuación se expondrán las pruebas y validaciones realizadas dentro del capítulo 7, y los mayores problemas y obstáculos hallados en la consecución del proyecto junto a la resolución tomada respecto a cada uno de ellos en el capítulo 8.

Finalmente se narrarán las conclusiones y las posibles líneas futuras de trabajo del proyecto en el capítulo 9.

2. Descripción general del sistema.

Como hemos mencionado antes, el proyecto se basa en un sistema de iluminación ya implementado en el B-105 Electronic Systems Lab. Dicho sistema cuenta con el tablón con las tiras de LED ya instaladas, la alimentación de las mismas mediante una fuente de 12V y un sistema de control, encabezado por un sistema empotrado Raspberry Pi.

2.1. Diagrama del diseño original.

En primer lugar, tenemos el diagrama hardware del diseño original, compuesto por los siguientes elementos:

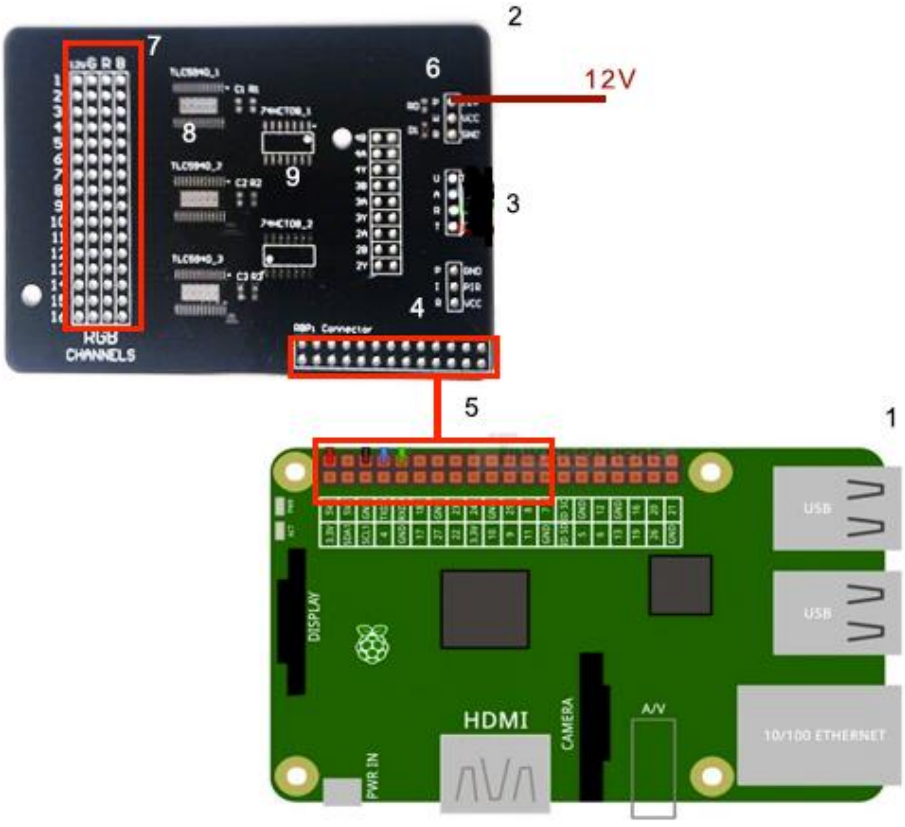

Ilustración 3: Diseño original del sistema de iluminación LED.

1) Raspberry Pi 2, modelo B.
2) Placa PCB Bulletin LED Board.
3) Conexión con el puerto serie de la Raspberry Pi.
4) Conexión del sensor de presencia.
5) Conexión de la PCB con la Raspberry Pi.
6) Alimentación a 12V y 5V de la PCB.
7) Salidas de los *drivers* TLC5940 hacia los LED.
8) *Drivers* TLC5940.
9) Puertas AND reguladoras de nivel de tensión.

Distinguimos dos elementos principales:

- El sistema empotrado Raspberry Pi, centro de todo el sistema de control (gestión de los pines de entrada/salida, UART y comunicación Bluetooth, así como la codificación de efectos).

- La placa PCB Bulletin LED Board, en la que se ubican los *drivers* TLC5940. La misión de estos drivers es enviar una señal PWM por cada una de sus salidas a las tiras de LED, configurando los diferentes colores.

En el próximo punto detallaremos cada uno de estos elementos de forma individual.

En cuanto al software, el diseño original contaba con una versión modificada de las librerías para TLC5940 de Raspberry Pi de código abierto que podemos encontrar en su página web [1]. El programa principal únicamente ejecutaba uno de los efectos previamente programados en el mismo hasta que el usuario lo terminase.

El diagrama del programa principal original se detalla en la siguiente página:

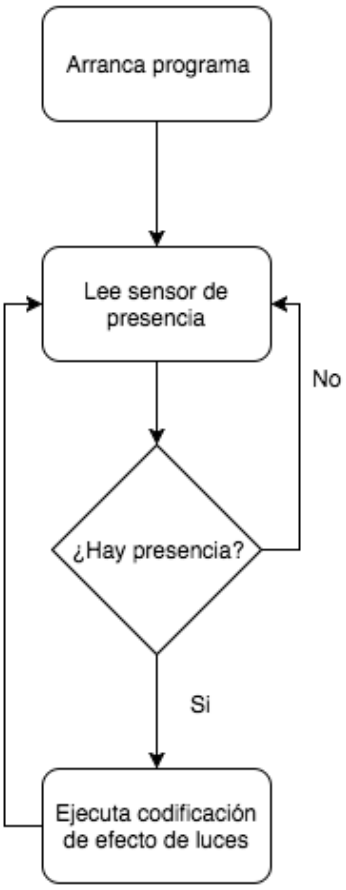

Figura 1. Diagrama del código original.

Dado que el sensor de presencia tenía como objetivo detectar a los transeúntes y consecuentemente activar el tablón, ha decidido prescindir de dicho sensor para la nueva implementación, dado que el tablón será activado por el propio usuario por su propia voluntad mediante el teléfono.

2.2. Definiciones.

Previamente a realizar la descripción del sistema, resulta interesante dedicar un apartado a aclarar varios de los términos que se utilizan en esta memoria referidos al diseño global del mismo:

a) Segmento: Unidad mínima de LEDs controlables por la Raspberry. Cada uno de los segmentos posee tres canales (rojo, verde y azul). Dentro de un segmento se puede regular el nivel de luminosidad de cada uno, pero no se pueden configurar diferentes niveles de un mismo color.

b) Canal: Representa cada una de las salidas de los TLC5940. Más concretamente, se trata de un color específico en un segmento específico. Por ejemplo, el rojo

del segmento 12. Hay 16 segmentos en total, con tres canales por cada uno, que hacen un total de 48 canales.

c) Nivel de luminosidad: Los TLC5940, como ya se ha mencionado, pueden establecer 4096 niveles de luminosidad diferentes en cada canal. Este nivel de luminosidad se realiza mediante una señal PWM y la combinación de distintos niveles de luminosidad en un mismo segmento es la base para configurar los distintos colores.

2.3. Hardware

En este punto se detallan todas las partes hardware del sistema previamente mencionadas.

a) Raspberry Pi 2 modelo B.

El sistema empotrado Raspberry Pi 2 modelo B es el sistema central de control del diseño hardware del que consta el proyecto. Dicha Raspberry cuenta con un microcontrolador Broadcom BCM2836 y sistema operativo Raspbian.

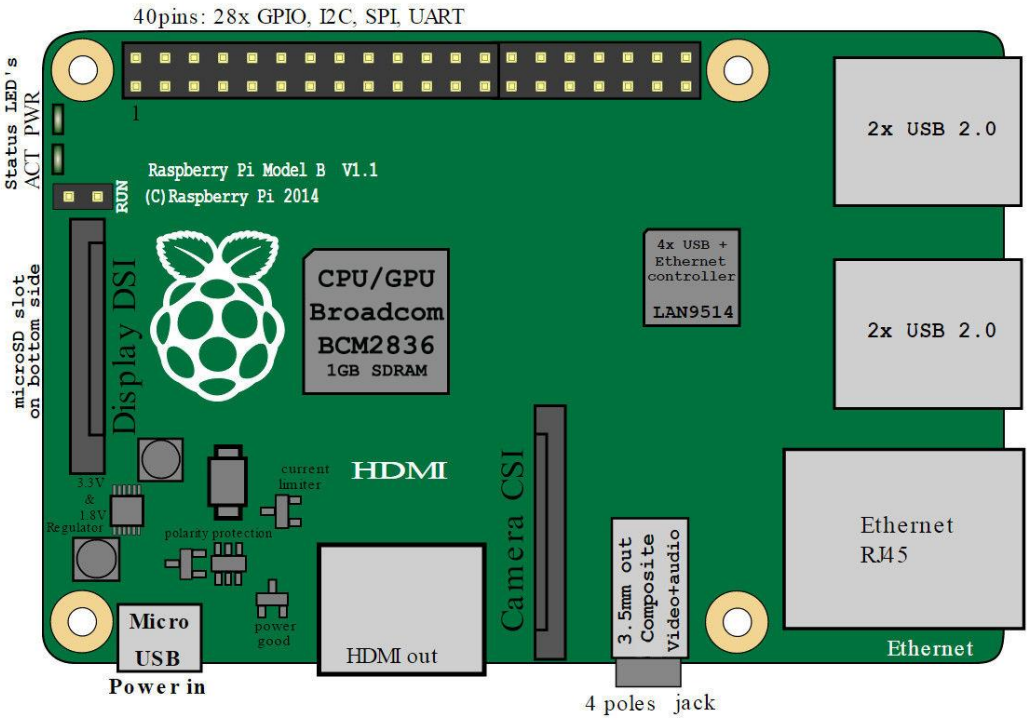

Ilustración 4. Raspberry Pi 2 modelo B.

Destacamos los 40 pines de entrada y salida de los que consta, entre los cuales se encuentra la UART del sistema, así como los GPIO que configuraremos para enviar a los TLC5940 las señales de control para activar las luces.

b) La placa PCB Bulletin LED Board.

La placa PCB Bulletin LED Board es un diseño anterior creado en el B-105 Electronic Systems Lab y gracias a la cual se simplifican las conexiones entre la Raspberry Pi y los *drivers* TLC5940, además de facilitar la salida hacia las tiras de LED. Todos los componentes del diseño son SMD

El diagrama de bloques de la placa Bulletin LED Board se muestra en la siguiente página.

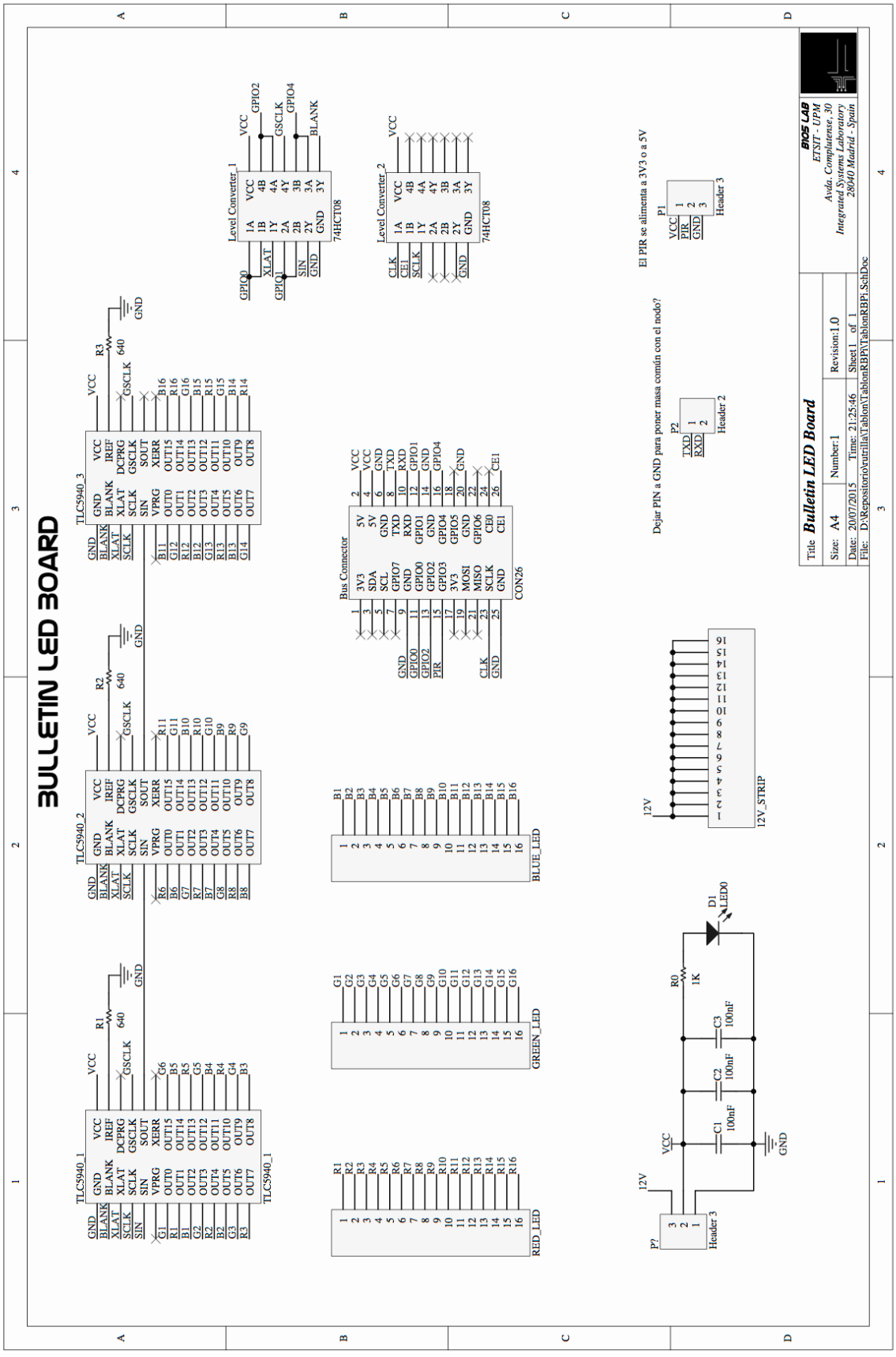

Ilustración 5. Esquemático de la placa PCB Bulletin LED Board [2]

Destacan en la placa los conectores para Raspberry Pi, las puertas AND (que actúan de reguladoras de nivel de tensión, convirtiendo las señales de 3,3V de los GPIO de Raspberry Pi a 5V) y los controladores TLC5940.

El *driver* analógico TLC5940 [3] es un circuito integrado de Texas Instruments con control de luminosidad de escala de grises mediante PWM. Posee 16 canales de salida de corriente constante con 4096 niveles diferentes de luminosidad cada uno. Además de ello cuenta con un sistema de corrección de luminosidad o circuitos de información de errores.

Se alimenta con una señal de entre 3 y 5,5 V. En nuestro caso utilizaremos una señal de 5V procedente de la Raspberry Pi.

Las señales de entrada que utiliza el TLC5940, si bien posee más señales para otras configuraciones no contempladas, son tres: Sin, GSclk y Sclk.

Los drivers pueden colocarse en cascada utilizando su salida Sout y utilizando el mismo reloj Sclk y GSclk para ellos. En nuestro caso tenemos un diseño con tres *drivers*, cada uno de los cuales posee 16 canales de salida, haciendo un total de 48 canales diferentes, repartidos en 16 tiras de LED (tres canales por tira).

2.4. Software.

El programa a cargar en la Raspberry, encargado del control de la placa Bulletin LED Board, está basado en la librería para TLC5940 de código abierto y en las modificaciones de esta librería que fueron realizadas en el laboratorio.

a) Librería TLC5940 para Raspberry Pi.

La librería TLC5940 para Raspberry Pi tiene como misión facilitar la codificación de efectos enviando las señales de control pertinentes a las entradas, simplemente indicando el valor de luminosidad que deseemos al canal correspondiente.

Para crear los diferentes efectos de luces para el tablón, es necesario controlar las distintas transiciones de los 48 canales. Para ello, hay que introducir los datos por una línea serie y validar la salida cuando esté lista.

Dentro del código de la librería, en el archivo *tlc_controller.h*, debemos diferenciar:

- Por un lado, la declaración de la clase *TLC_Controller* para poder crear una instancia por cada TLC5940 a controlar, así como obtener el control de todos los pines de Raspberry Pi a los que está conectado.

- La función ***update()***, encargada de mandar la información que hemos programado al TLC5940. Debe ser llamada constantemente para actualizar las señales enviadas.

  Es importante resaltar que si la función ***update()*** no es llamada constantemente, no tendremos a la salida una señal estable de PWM. Esta función toma los valores de luminosidad que hemos escogido previamente para enviar las señales pertinentes al TLC5940.

- La definición tipo *color_t* es un entero sin signo de 16 bits que representa la luminosidad que se aplicará a un canal determinado.

- La variable *colors* es un array de *color_t* de tamaño NUM*16, siendo NUM una constante que representa el número de instancias de TLC5940 que hemos invocado.

- La función ***void setChannel(unsigned int* channel*, color_t* value*)*** rellena la posición *channel* del array *colors* con el valor *value*. Esto es, asigna a uno de los canales un color determinado.

- La función ***void setAll(color_t* value*)*** rellena todos los canales con un valor de luminosidad concreto.

- La función ***void clear()*** rellena todo el array *colors* con ceros. En definitiva, apaga todas las luces.

- Funciones propias para la codificación de los diferentes efectos, que serán detalladas en el próximo punto.

b) La clase Color.

Además de los mencionados ficheros utilizados para el control de las salidas de los TLC5940, existe una clase auxiliar Color en el fichero *color.cpp* que representa colores. Pueden extraerse los componentes RGB de una instancia de la clase Color mediante sus atributos.

Por ejemplo, el color morado tiene un valor de 0x00FFFF [4]. A la hora de extraer cada una de las componentes, se aplica una máscara que permite extraer el valor de cada una de las componentes de ese color. En este caso, obtendríamos un valor de 255 para azul, 255 para rojo y 0 para verde.

3. Diseño modular del sistema.

En este capítulo se detallan las variaciones introducidas respecto al sistema original. En primer lugar se expone un diagrama con las novedades físicas introducidas, es decir, tanto el módulo Bluetooth elegido para la Raspberry Pi 2 como el dispositivo móvil desde el cual se controlará el mismo por Bluetooth.

Con estos añadidos, se consigue, en primer lugar, una interacción usuario-tablón inmediata de cara a personalizar diferentes efectos de luces; en segundo lugar, se consigue eliminar la necesidad de reescribir el código en la Raspberry Pi cada vez que se quiera cambiar el efecto, por lo que cualquier persona puede cambiar los efectos de forma sencilla sin necesidad de conocimientos de programación.

3.1. Diagrama del nuevo diseño.

Nuestro nuevo diseño, por tanto, queda de la siguiente forma:

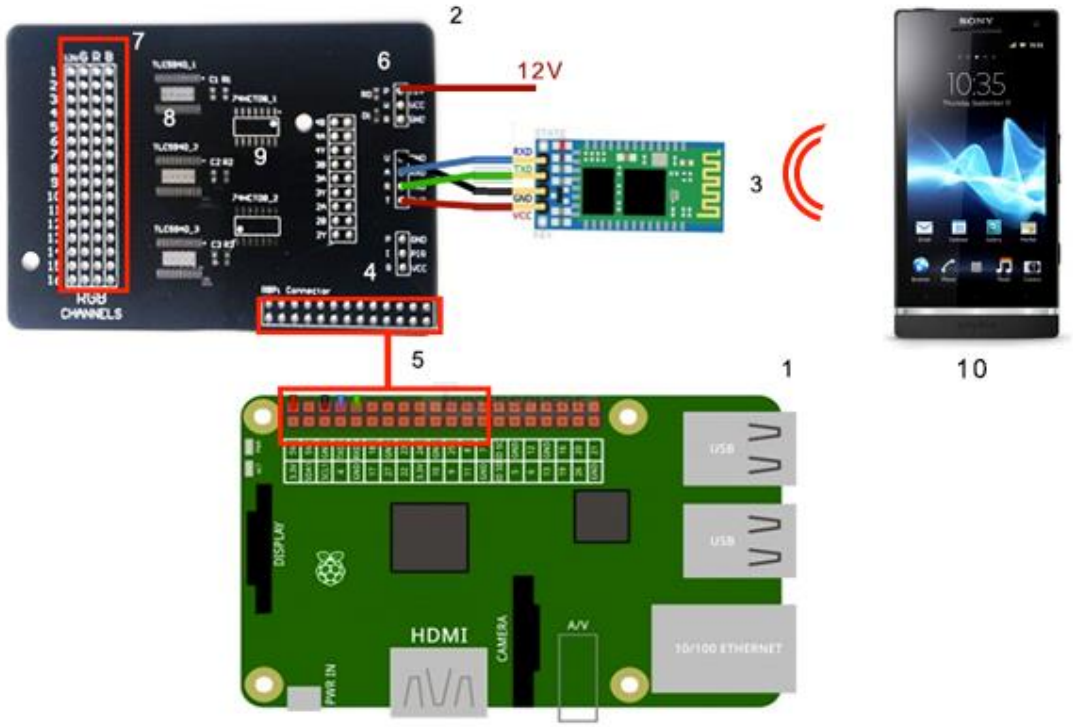

Ilustración 6: Nueva configuración modular del sistema.

1) Raspberry Pi 2, modelo B.

2) Placa PCB Bulletin LED Board.
3) Módulo HC-05.
4) Conexión del sensor de presencia (no utilizado).
5) Conectores de la PCB y sus correspondientes pines en la Raspberry Pi.
6) Alimentación a 12V y 5V de la PCB.
7) Salidas de los TLC5940 hacia los LED.
8) *Drivers* TLC5940.
9) Puertas AND reguladoras de nivel de tensión.
10) Teléfono inteligente con sistema Android.

Respecto a la elección del sistema Android, que será detallada en el capítulo 5 de esta memoria, existen una gran variedad de recursos como librerías, referencias o ejemplos, tanto proporcionados por Google como de terceros, que facilitan el desarrollo de cualquier aplicación, por lo que se ha considerado una opción interesante a la hora de elegir una plataforma móvil.

3.2. El módulo Bluetooth.

Se ha decidido utilizar el estándar Bluetooth para la comunicación entre la Raspberry y los dispositivos móviles, dado que garantiza la posibilidad de usarlo por la práctica totalidad de teléfonos que se comercializan actualmente, así como la sencillez de uso y a la hora de escribir el código y la lógica que gestione la comunicación.

Para poder obtener conectividad Bluetooth entre la Raspberry Pi y otros dispositivos se ha optado por la utilización de un módulo Bluetooth SPP, el módulo HC-05 [5], diseñado para una configuración de conexión serie transparente. Este módulo está diseñado para funcionar de forma autónoma, por lo que únicamente debemos configurar el puerto serie para poder hacer uso del mismo, facilitando el desarrollo de aplicaciones que hagan uso del mismo.

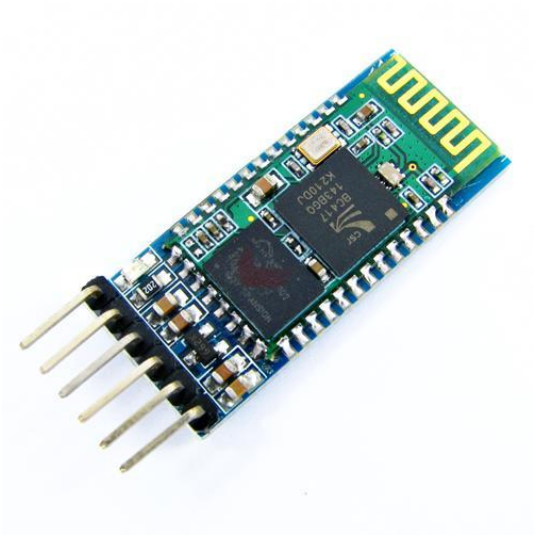

Ilustración 7. El módulo Bluetooth HC-05

Este módulo incorpora el protocolo Bluetooth 2.0, modulación de 3 Mbps con un transceptor de 2.4GHz. Consta también de una interfaz UART configurable por consola.

El módulo HC-05 se alimenta a una tensión entre 3 y 5,5V, y la lógica de su puerto serie tiene un nivel de 3,3V, el mismo utilizado por la Raspberry Pi, por lo que la conexión es directa, sin necesidad de regular el nivel de tensión de las señales intercambiadas entre ambos puertos serie.

El dispositivo puede configurarse en modo maestro o esclavo. De forma predeterminada, está configurado para actuar en modo maestro, emparejándose de forma sencilla con los dispositivos que lo requieran y sean conocedores de la contraseña asociada. Para facilitar el desarrollo de la app y el emparejamiento, dejaremos el dispositivo en modo maestro.

Se ha elegido este módulo por la sencillez que posee a la hora de realizar el descubrimiento, emparejamiento y conexión por Bluetooth con otros dispositivos, así como la comunicación serie que nos permite enviar una trama simple, suficiente para nuestra conectividad con un dispositivo Android, que veremos en capítulos posteriores.

3.3. Diseño del modelo de datos.

Para facilitar tanto la transmisión desde el dispositivo Android como la recepción en la Raspberry Pi, se ha decidido utilizar una trama de datos formada por un mensaje (*string*).

La trama tiene la siguiente forma. Cada uno de los campos está representado por un carácter dentro de la misma:

```
[  EFFECT | COLOR1 | COLOR2 | DURATION | STEP | OPTIONS ]
```

Se han elegido estos campos porque representan los parámetros de cada una de las diferentes funcionalidades que vamos a ejecutar en la Raspberry Pi. Dichas funcionalidades son diferentes efectos de luces que poseen una serie de parámetros (tipo de efecto, colores, duración…) y se configuran de diferente manera.

Cada uno de los caracteres que forman la trama representa una elección del usuario en cuanto a las opciones para configurar los efectos de luces del tablón. Se ha creado una tabla de opciones referente a cada uno de ellos:

- Effect: Tipo de efecto.

Carácter	Efecto
'a'	Blink
'b'	Intercalar
'c'	Dimmer

- Color1 / Color2**:** Colores determinados para los efectos.

Carácter	Color
'r'	Rojo
'g'	Verde
'b'	Azul
'y'	Amarillo
'w'	Blanco
'p'	Morado
'o'	Naranja

- Duration: Duración del efecto deseado.

Carácter	Duración
'1'	100 ms
'2'	250 ms
'3'	500 ms
'4'	1 s
'5'	2 s

- Step y Options dependen del efecto que queramos configurar y representan el paso entre niveles lumínicos y posibles opciones para futuros efectos a configurar.

La trama es ampliable de forma muy sencilla hasta la longitud que se desee. Únicamente debemos extender el buffer de recepción hasta el límite que deseemos y utilizar los caracteres como símbolos para representar un valor de parámetros.

Resulta sencillo también ampliar las diferentes opciones de los diferentes caracteres, solo hay que incluirlos en el programa e implementarlos de manera similar a los anteriores.

4. Desarrollo de funcionalidades en Raspberry Pi.

Una vez hemos diseñado e implementado nuestro nuevo sistema, pasamos a comentar el diseño software del mismo. A lo largo de las próximas páginas se detallará todo el desarrollo software relacionado con el código que se ejecutará en Raspberry Pi.

En primer lugar, comentaremos la función que realiza el programa principal, llamado *main.cpp*. En él hallamos el código referente al ejecutable, en el cual se llama a las funciones correspondientes a cada efecto de luces deseado y se gestiona la comunicación serie con el módulo HC-05. Consta de dos hilos de ejecución claramente diferenciados, el primero de ellos encargado de la recepción de datos por el puerto serie y el segundo, cuya misión es llamar a los diferentes efectos personalizados en función de los parámetros recibidos.

A continuación, conoceremos en profundidad todo lo relacionado con la conectividad Bluetooth desde el punto de vista de la Raspberry Pi y el uso del módulo HC-05 para tal fin.

Por último, se expondrán con mayor nivel de detalle cada una de las funciones implementadas para la codificación de los diferentes efectos.

4.1. Programa principal.

Como ya hemos mencionado, el programa está desarrollado en el lenguaje C++ y está basado en el uso de la librería TLC5940.

Este código incluye las librerías *thread, wiringPi* [6], los ficheros relativos al puerto serie y el fichero auxiliar *efectos.cpp*:

- *thread*: Utilizamos la librería thread para crear dos hilos de ejecución en paralelo: uno para la ejecución de los diversos efectos y otro para la gestión del puerto serie.

- *wiringPi*: Se trata de una librería para acceder a los pines GPIO BCM2835 de la Raspberry Pi escrita en C, bajo la licencia GNU LGPLv3.

  La librería wiringPi es necesaria para acceder de forma sencilla a los pines GPIO de la Raspberry. El uso de la librería TLC5940 exige el uso de wiringPi.

- Librerías para el manejo del puerto serie: se trata de una serie de librerías de C que se utilizan para realizar diversas acciones relacionadas con el puerto serie: *stdio.h, fcntl.h* y *termios.h.*

- Librería TLC5940: Como ya hemos mencionado, haremos uso de la librería disponible para Raspberry Pi de los TLC5940.

a) Diagrama de la ejecución.

El diagrama de la ejecución principal, que se detallará a lo largo de los próximos puntos, es el siguiente:

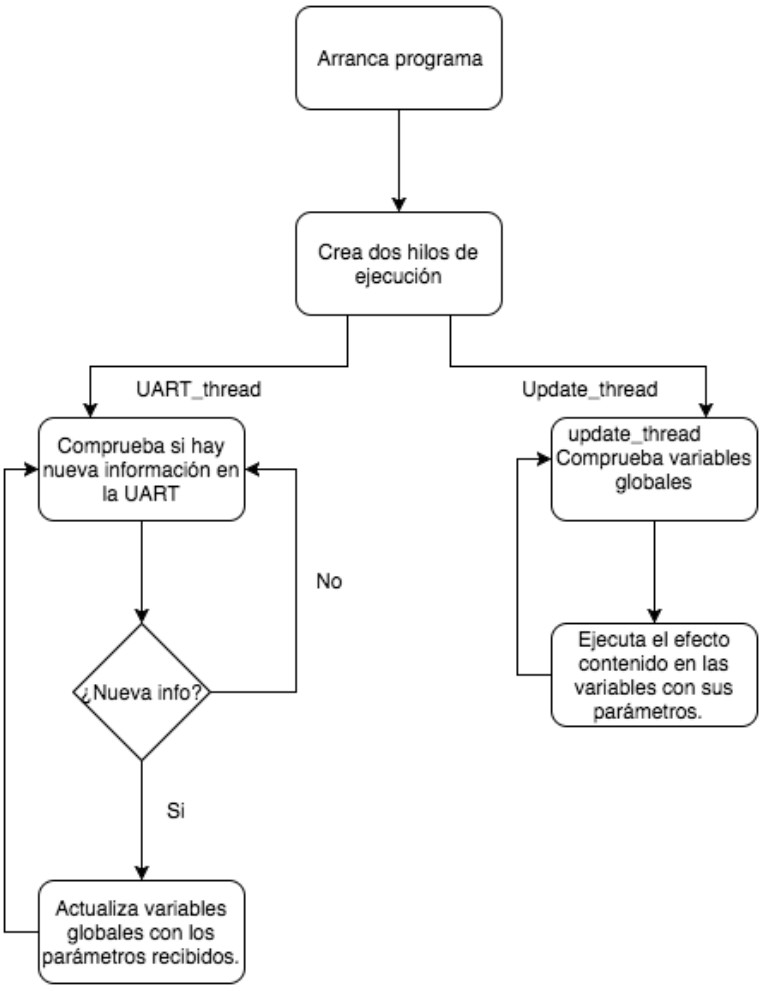

Figura 2. Diagrama del programa principal.

Como podemos observar, dentro del programa principal existen varias variables de control. Existen dos hilos de ejecución diferenciados que llevan a cabo la actualización y lectura de variables, que procedemos a detallar a continuación.

b) Hilos de ejecución.

El programa principal consta de dos hilos de ejecución claramente diferenciados, y ***UART_thread()***:

- ***UART_thread()*** se encarga de la recepción de datos por el puerto serie. Una vez recibida la trama, se almacena en el buffer creado a efecto de guardar la información recibida. Después, cada carácter de la trama se guarda en una variable que podrá ser leída en el anterior hilo de ejecución.

- ***update_thread()*** es el hilo de ejecución relativo a la codificación de los efectos. En él se encuentra toda la codificación de los efectos del tablón a partir de los parámetros recibidos por el puerto serie.

  Este hilo de ejecución lee las variables del programa en las que se escriben los datos recibidos desde el puerto serie y únicamente ejecuta el efecto que hemos enviado desde la aplicación con los parámetros de configuración elegidos.

4.2. Comunicación Bluetooth.

Tal y como se ha mencionado anteriormente, para la comunicación por Bluetooth con el dispositivo Android, se ha optado por el módulo HC-05. El módulo es prácticamente autónomo y únicamente necesita del puerto serie para comunicarse por Bluetooth.

a) Configuración del módulo HC-05

En primer lugar, describiremos la configuración del puerto serie para la comunicación por la UART. Dicha configuración tiene que ser suficiente para que se reciban correctamente los datos enviados desde el teléfono.

- Baudios: 9600. No es necesaria una velocidad alta.
- Trama: 8 bits de datos.
- Bit de paridad: no.
- Bits de parada: 1 bit.

Se ha elegido esta configuración debido a la sencillez y compatibilidad que proporciona a la hora de establecer la comunicación con otros transceptores Bluetooth.

b) Comunicación por el puerto serie.

Respecto a la comunicación por el puerto serie, se ha elegido utilizar las librerías en C de edición de ficheros *fcntl.h* [7]. El *stream* de datos utiliza como fichero el relativo al puerto serie de la Raspberry Pi, situado en */dev/ttyAMA0*. Inicialmente abrimos una conexión con el puerto serie.

```
uart0_filestream = open ("dev/ttyAMA0", O_RDWR | O_NOCTTY | O_NDELAY);
```

Con esta sentencia abrimos un flujo de datos (*stream*) con la UART. Después se configura el puerto serie con los parámetros ya comentados.

Es importante destacar que, si bien estamos en un modo de no bloqueo y podríamos ubicar el código referente a la recepción de datos en el mismo hilo que la gestión de los LEDs, es precisamente por no bloquear la ejecución de éste último que ha de ser constante, por lo que debemos crear un nuevo hilo de ejecución en el cual coloquemos la parte del código referente a la recepción por Bluetooth de los datos.

Una vez tengamos configurada la UART, el siguiente paso es escribir el código referente a la recepción de datos por el puerto serie como tal. Como hemos dicho, para ello usaremos el hilo ***UART_thread().***

- En primer lugar, creamos el *buffer* de recepción.

- Después leemos del archivo del puerto serie, utilizando el *buffer* previamente abierto y como máximo leemos 255 bytes. El tamaño del *buffer* es guardado en una variable rx_length, para posteriormente colocar al final un carácter '\0' que indica fin de transmisión.

```
unsigned char rx_buffer[256];
int rx_length = read(uart0_filestream, (void*)rx_buffer, 255);
```

Una vez almacenados los datos de lectura en el *buffer*, comprobamos su tamaño con la variable rx*_buffer,* que guarda el valor devuelto por la función ***read()*** :

- Si la función ha devuelto un valor de -1, se ha producido un error que debemos tratar.

- Si ha devuelto un valor de 0, no tenemos datos esperando.

- Si devuelve un valor mayor que 0, entonces colocamos el carácter de terminación y finalmente guardamos los valores de la trama en sus respectivas variables del programa local: efecto, color, duración, etc. En el próximo punto detallaremos los puntos clave de la trama.

4.3. Funciones y codificación de los diferentes efectos.

Una vez se han recibido los datos por el puerto serie mediante las funciones expuestas en el punto anterior, pasamos a hacer uso de los parámetros recibidos en el hilo correspondiente a la gestión del puerto serie.

Las diferentes funciones de efectos son llamadas desde el hilo de ejecución ***update_thread()*** del programa principal, si bien el código de cada una de las diferentes funciones está escrito en el fichero auxiliar *efectos.cpp*, como hemos señalado en el primer punto de esta sección.

Las funciones de los efectos tienen la siguiente estructura:

```
template<unsigned int NUM>
void TLCController<NUM>::blink(Color color, int segment){        // (1)

      for (k = 0; k < 16; k++){                                   // (2)
            this->setChannel(k*3, color.g);
            this->setChannel((k*3) + 1, color.r);
            this->setChannel((k*3) + 2, color.b);
      }

      for(j=0;j<500;j++){                                         // (3)
            this->update();
      }

      //… resto de la función
}
```

1) Nombre de la función y parámetros. Consideraremos nombres que hagan referencia al comportamiento del efecto a implementar.

2) Cuerpo de la función. Utilizamos bucles for para recorrer todos los canales o segmentos que queramos incluir.

3) Llamada a la función ***update()*** en bucle, según la duración que queramos de un color determinado. Aproximadamente cada iteración del bucle corresponde a 1 ms.

En general, este es el esquema seguido a la hora de crear las diferentes funciones y efectos para el tablón. Para esta memoria se han implementado los siguientes:

- ***void on_segment_timed(Color* color*, int* segment*, int* duration*):*** Coloca un color determinado en el segmento que elijamos durante el periodo de tiempo que deseemos (*duration*, en ms).

  En caso de querer poner diferentes colores de forma secuencial solamente tenemos que concatenar ésta función con diferentes colores e intervalos de duración. Puede utilizarse como base para crear otras funciones similares más complejas.

- ***void blink(Color* color*):*** hace parpadear el tablón con el color que elijamos. Enciende todos los leds de ese color durante 500 ms y los apaga durante el mismo periodo de tiempo.

- ***void intercalar(Color color1, int duracion1, Color color2, int duracion2):*** intercala dos colores de forma consecutiva con la duración que elijamos para cada uno.

Estos efectos pueden combinarse de diferentes formas para configurar secuencias más complejas cuyos parámetros serán enviados desde el dispositivo Android. Además, se han utilizado también efectos que ya se encontraban implementados anteriormente en el laboratorio.

- ***void dimmerUpAll(int* r*, int* g*, int* b*, unsigned int* duration*):*** crea un efecto de degradado, de menos a más luminosidad, en todo el tablón.

- ***void dimmerDownAll(int* r*, int* g*, int* b*, unsigned int* duration*):*** similar al anterior, pero disminuyendo la luminosidad.

5. Aplicación Android.

Uno de los objetivos del proyecto, como hemos mencionado, es poder controlar desde un dispositivo móvil los diferentes efectos de luces para así personalizar de forma sencilla e intuitiva el tablón expositor.

Se ha decidido desarrollar la aplicación para la plataforma Android por las siguientes razones:

- Android es un sistema de código abierto, lo cual facilita al desarrollador la libertad para crear y publicar su aplicación de forma completamente gratuita. Además existe una gran variedad de recursos como librerías, referencias o ejemplos, tanto proporcionados por Google como de terceros, que facilitan el desarrollo de cualquier aplicación.

- Actualmente la cuota de mercado de Android es de un 84,1% [8], por lo que alcanzaremos a una mayor cantidad de sistemas que podrían usarlo.

- Su versatilidad permite adaptar el código fácilmente a los diferentes sistemas que utilizan Android (*tablets, wearables*…), así como la accesibilidad para continuar desarrollando en el futuro nuevas funciones para el tablón u otros sistemas inalámbricos del laboratorio.

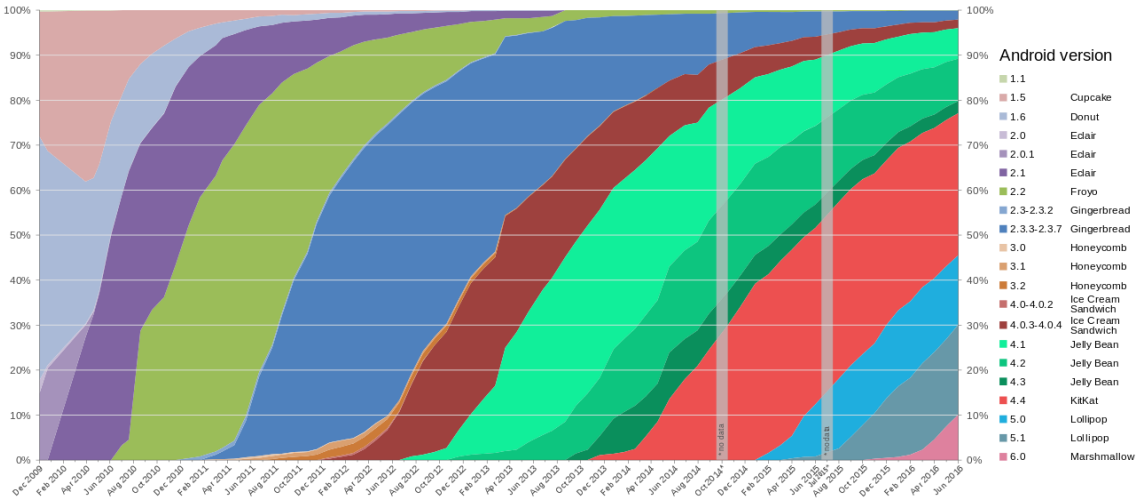

Ilustración 8: Detalle de la cuota de mercado de las diferentes versiones de Android.

5.1. Características.

- **Nombre de la aplicación:** Bulletin LED Board.
- **Nivel mínimo de API:** 16 (Jelly Bean).
- **Nivel objetivo de API:** 23 (Marshmallow).
- **Tamaño de la aplicación:** 5,12 MB.

Buscando alcanzar los puntos y objetivos mencionados, se ha optado por elegir un nivel mínimo de API de 16, correspondiente a la versión 4.1 de Android, *Jelly Bean*. Se trata de una versión muy estable lanzada en julio de 2012 que supuso un aumento significativo en la optimización y rendimiento del sistema. Más de un 95% de los sistemas Android soportan actualmente un nivel de API igual o mayor a 16 por lo que el alcance es más que suficiente.

Por otro lado, establecemos un nivel objetivo de API de 23, que se corresponde con la última versión Android 6 *Marshmallow*. Una de las ventajas del sistema es poder desarrollar el software para la última versión, aprovechando las novedades y ventajas que posee, y desactivar de forma automática las funciones que no sean compatibles con la versión de Android en la que ejecutemos la aplicación. Android 6 incluye un gestor de consumo energético muy avanzado y una nueva interfaz de usuario.

Dentro del fichero *AndroidManifest.xml* se declaran tanto los niveles de API mínimo y objetivo como los permisos de Bluetooth necesarios para que la aplicación pueda acceder al hardware necesario.

También se declaran, asimismo, las diferentes actividades (*Activity*) de las que consta la aplicación, y que serán detalladas en los próximos puntos.

5.2. Estructura y diagrama de la aplicación.

Dependiendo de la forma en la que estén programadas las diversas actividades, el paso de una a otra pantalla de actividad puede realizarse de forma automática o bien esperando una interacción del usuario.

El diagrama de la aplicación es el siguiente. Cabe destacar que cada actividad posee su propio botón de idéntico nombre, y volver atrás se refiere al uso del botón pertinente en el teléfono:

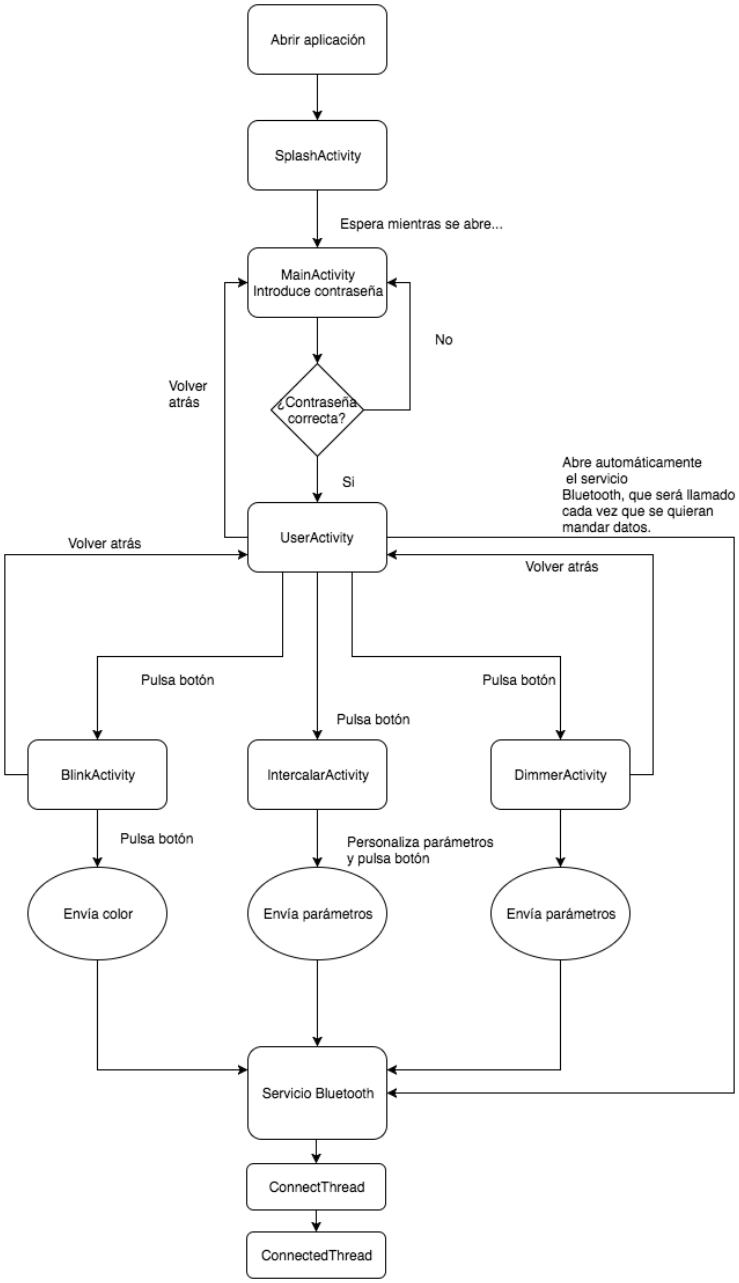

Figura 3. Diagrama de la aplicación Android.

- En primer lugar, una pantalla de presentación o *SplashScreen*, con el logo del B-105 que aparece durante unos segundos como bienvenida.

- A continuación, la actividad *MainActivity*, que requiere las credenciales de usuario. En principio bastará con conocer una contraseña básica y será accesible a cualquiera que la conozca.

- La actividad central, *UserActivity*, es un menú de selección de la funcionalidad del tablón y también a las diferentes opciones de configuración y personalización que proporciona la aplicación.

- Cada una de las diferentes funcionalidades para el tablón tiene su propia actividad configurable. Es en estas actividades donde se puede personalizar el efecto deseado y se lleva a cabo el envío de la información por Bluetooth.

Por otro lado, para gestionar la comunicación Bluetooth, se ha implementado un servicio Bluetooth que será detallado con precisión posteriormente. Este servicio se ejecuta en segundo plano sin interacción alguna con el usuario y no se detiene al cambiar de actividad. En el apartado 5.4 serán explicadas con mayor detalle.

5.3. Definiciones.

Llegados a este punto, resulta interesante exponer algunas definiciones de términos dentro del mundo de Android:

- Actividad (*Activity*): Una aplicación en Android se compone de diferentes actividades. En general, se trata de las diferentes pantallas de las que dispone una aplicación. Está formada por una parte lógica, una clase .java que extiende de la clase Activity, y una parte gráfica, que está representada por un archivo XML. El usuario interactúa con las diferentes actividades mediante la interfaz gráfica.

- Servicio (*Service*): Se trata de un elemento que lleva a cabo operaciones de larga duración en segundo plano, sin necesidad de interactuar con el usuario, y que no dispone por tanto de una interfaz gráfica.

- *Layout*: Es la organización, declarada de forma expícita, de los diferentes elementos de una interfaz gráfica (botones, cuadros de texto, selectores…).

5.4. Diseño de la interfaz de usuario.

La interfaz de usuario de la aplicación pretende ser sencilla y accesible a todo usuario que quiera hacer uso de la misma. Dispone de varias pantallas (*Activity*) que siguen el diagrama anteriormente expuesto.

a) Actividades.

Las actividades que componen la aplicación son las siguientes:

- *SplashActivity.*

Se trata de la pantalla de bienvenida de la aplicación. Más allá de la mera estética, la pantalla sirve como espera mientras termina de cargar la aplicación, por lo que inmediatamente se lanza la actividad *MainActivity* a continuación.

La pantalla no dispone de *layout* propio, únicamente dispone de un *background* personalizado con el logo del B-105.

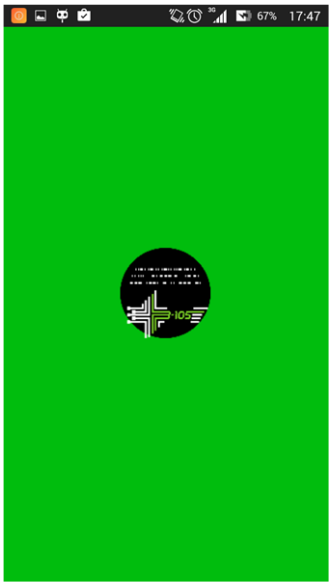

Ilustración 9: Pantalla de SplashActivity.

- *MainActivity.*

Se trata de la actividad inicial de la aplicación. Se nos pide en ella un usuario y contraseña para acceder al resto de funcionalidades de la aplicación.

Como simplificación, la contraseña no será más que un campo de texto que se comparará con una constante preestablecida.

Ilustración 10. Pantalla de MainActivity.

- *UserActivity.*

La actividad de usuario es la actividad central de la app, utilizada como menú principal para acceder a las distintas funcionalidades de la misma. Desde *UserActivity* podemos seleccionar la función del tablón deseada para poder configurarla según deseemos.

Se incluye también una opción de test de Bluetooth para comprobar si está conectado o no el dispositivo con el módulo HC-05.

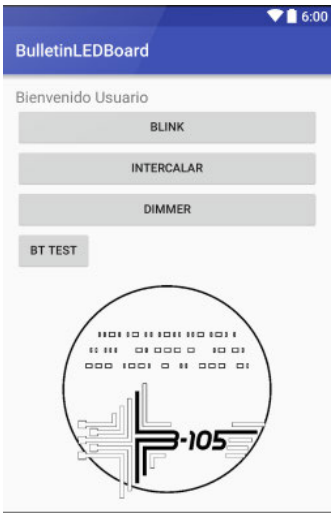

Ilustración 11. Pantalla de UserActivity.

Desde esta actividad también se realizan varias operaciones referentes a la utilización del Bluetooth:

- En primer lugar, se consulta al usuario si desea activar y permitir la conectividad Bluetooth del teléfono, en caso de que no esté ya activado.

- Si el Bluetooth está activado o bien se activa aceptando el primer punto, se abre el servicio Bluetooth que gestionará todo lo referente a la conectividad Bluetooth.

- *BlinkActivity.*

BlinkActivity es la actividad que envía los parámetros del código relacionado con el efecto ***blink()*** en la Raspberry Pi. Se accede pulsando el botón *BLINK* en UserActivity.

Dado que este efecto es el más básico y únicamente hace parpadear el tablón con un color que elijamos desde el Smartphone solamente contiene varios botones con los que podemos elegir el color.

- *IntercalarActivity.*

De forma homóloga tenemos la actividad *IntercalarActivity*, la cual llama al efecto ***intercalar().*** Se accede mediante el botón *INTERCALAR.*

Para configurar este efecto tenemos que enviar ciertos parámetros, como son la duración de cada color y qué colores van a intercalarse consecutivamente.

- *DimmerActivity.*

DimmerActivity llama a los efectos *DimmerUpAll()* o *DimmerDownAll()* según los campos de color, selección de degradado ascendente o descendente y duración del degradado.

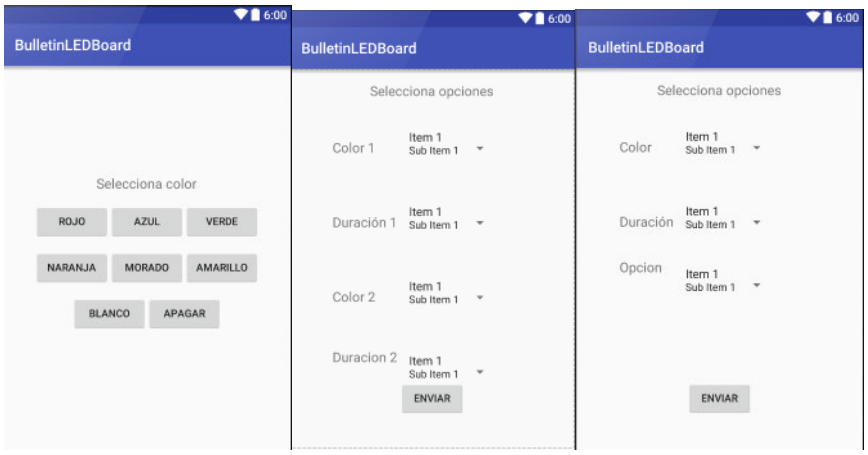

Ilustración 12: Las diferentes actividades de configuración de efectos.

Además de las diferentes pantallas de actividad, se ha incluido un estilo personalizado que incluye un fondo de pantalla con el logo del B-105. Este estilo se incluye dentro del fichero *styles.xml* en la carpeta *res* de la aplicación. También se ha sustituido el icono de la app por el del laboratorio.

5.4. Bluetooth y clases auxiliares.

Respecto a la gestión de la conectividad Bluetooth por parte de la aplicación Android, se ha hecho uso de las librerías que proporciona Google a tal efecto que simplifican en cuantía el proceso. Además se han creado una serie de clases auxiliares, según las directrices de Android, encargadas de gestionar el mismo.

Podemos crear una conexión utilizando la dirección MAC del módulo HC-05 y un identificador universal (UUID) para la susodicha conexión. Para la gestión de dicha conexión, hemos creado un servicio dentro de la aplicación Android.

Utilizando un servicio para la gestión del Bluetooth, en el que abrimos los hilos encargados de crear y mantener la conexión, así como el hilo por el que enviamos los datos mediante el OutputStream, conseguimos mantener nuestra conexión Bluetooth aún cambiando de actividad dentro de la propia aplicación, dado que en segundo plano sigue ejecutándose la tarea adscrita a la misma.

Debido a que necesitamos poder cambiar entre actividades de forma rápida y dinámica sin perder la conexión Bluetooth para poder cambiar y personalizar los diferentes efectos de luces, el servicio nos garantiza una fiabilidad suficiente para cumplir los requisitos que pide el diseño.

Para todo ello ha empleado la API de Bluetooth de Google [9], la cual consta de una serie de librerías que simplifican el proceso de conectado. Debido a que la interacción con el módulo HC-05 es muy sencilla, dado que este módulo crea de forma automática una conexión con los dispositivos con los que emparejamos el dispositivo, utilizaremos el teléfono en modo cliente, que resulta en una forma sencilla de establecer comunicación.

a) Diagrama de la conectividad Bluetooth.

De cara a gestionar la conectividad Bluetooth, se ha creado un servicio, una parte de código que se ejecutará en segundo plano y en la cual se llamarán a los hilos de ejecución que tienen como tarea crear y gestionar una conexión con el módulo HC-05. Cada uno de estos elementos será detallado en los próximos puntos.

El diagrama de la ejecución es el siguiente:

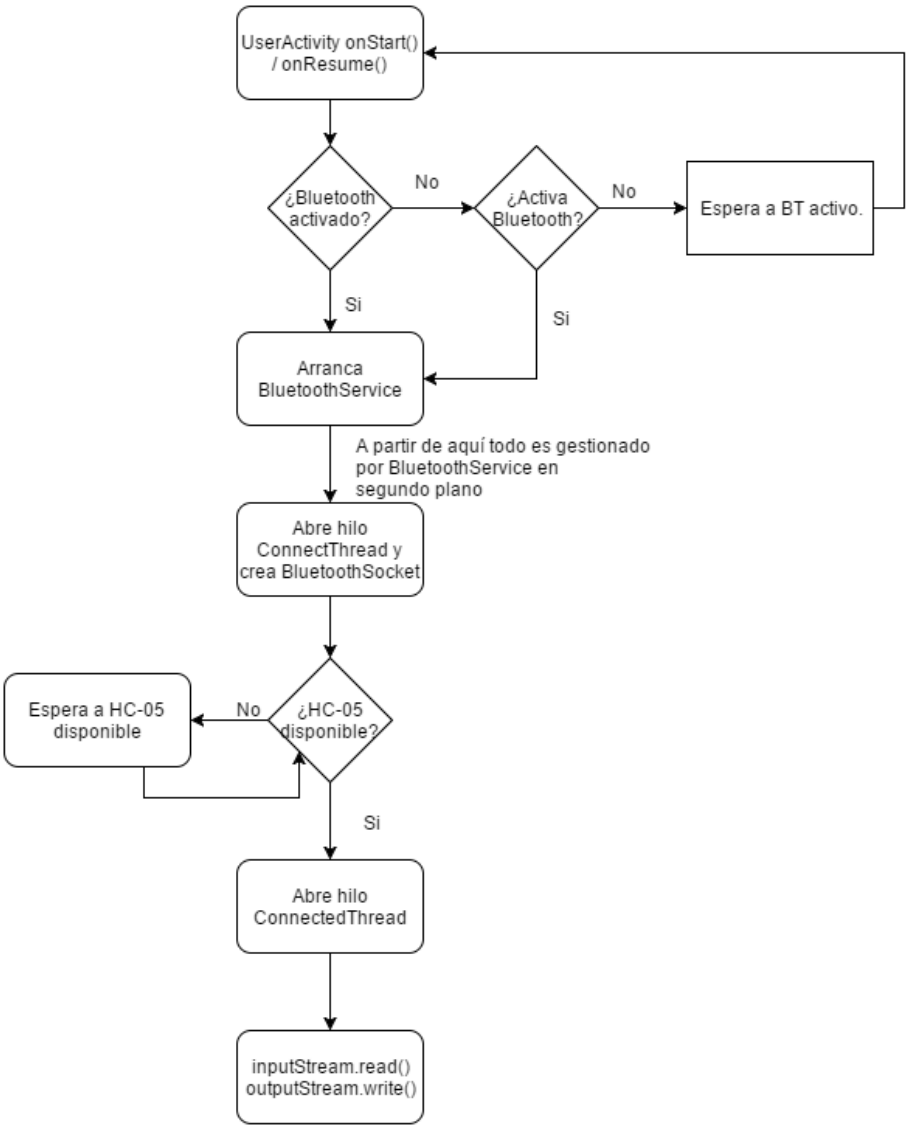

Figura 4. Diagrama Bluetooth de la aplicación.

b) El servicio BluetoothService.java.

Se ha creado un servicio de Android encargado de la gestión de la conexión Bluetooth. Como hemos dicho anteriormente, se trata de un elemento sin interfaz gráfica que nos permitirá manejar la conexión Bluetooth en segundo plano.

```
private final IBinder mBinder = new LocalBinder();                    (1)

public static String address = "98:D3:31:B3:9C:48";                   (2)

public BluetoothAdapter btAdapter;                                    (3)
public BluetoothDevice device;                                        (4)
public ConnectThread conexion;                                        (5)
```

1) Se crea una instancia de IBinder. En general, la clase Binder es la encargada de gestionar la conectividad entre servicios ligados a actividades (*Bound Services*) y las diferentes actividades que necesitan hacer uso del mismo.

2) *BluetoothDevice* representa una instancia del dispositivo al que vamos a conectarnos. Podemos acceder al dispositivo pasando por parámetro la dirección MAC única del mismo.

3) La constante *address*, en la que guardamos la MAC del dispositivo HC-05, dado que es el único al que vamos a conectarnos.

4) Creamos un hilo de ejecución nuevo *ConnectThread* para abrir una conexión con el dispositivo externo. Esto es debido a que crear una conexión se realiza mediante un método que bloquea el hilo de ejecución hasta que se realice.

En el constructor de la clase, por tanto, asignamos valores a estas constantes e iniciamos el hilo de ejecución de *ConnectThread*.

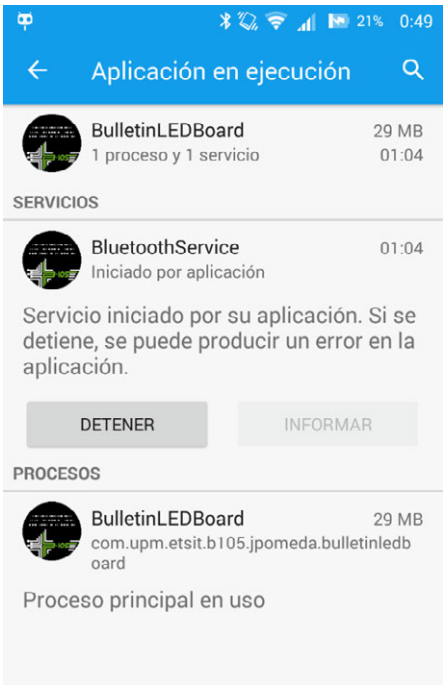

Ilustración 13: BluetoothService en ejecución

c) ConnectThread.java.

Se trata de un hilo de ejecución de la aplicación a la cual pasamos un BluetoothDevice, ya explicado, y nos crea un *BluetoothSocket,* punto de intercambio de información desde la aplicación hacia el módulo Bluetooth. De la clase ConnectThread destacamos los siguientes atributos:

```
private final BluetoothSocket mmSocket;                    (1)
private final BluetoothDevice mmDevice;

ConnectedThread mConnectedThread;                          (2)
```

```
private final static UUID MY_UUID = UUID.fromString("00001101-0000-
1000-8000-00805F9B34FB");                                          (3)
```

1) En primer lugar, utilizamos un *BluetoothSocket* que representa un punto de conexión que permite intercambiar datos con otro *BluetoothDevice* que habremos pasado a esta clase por parámetro, mediante *InputStream* y *OutputStream.* Es similar a la conexión que se realiza en TCP mediante sockets.

2) Por otro lado, creamos un hilo de ejecución más ConnectedThread, que será el encargado de gestionar los datos que se reciben y se envían por el mencionado *BluetoothSocket*.

3) Creamos una constante MY_UUID que guardará el identificador único universal (por sus siglas en inglés *Universally Unique Identifier*) asociado a la conexión que vamos a establecer. Concretamente, este valor es el estándar requerido para una conexión SPP (*Serial Port Profile*), dado que vamos a comunicarnos con un módulo Bluetooth que utiliza una interfaz serie.

En la parte del código referente al constructor, instanciamos el *BluetoothSocket* utilizando el método ***createInsecureRfcommSocketToServiceRecord(UUID uuid);*** llamado desde la instancia del *BluetoothDevice* el cual abrirá una conexión insegura con el dispositivo que hayamos pasado por parámetro al hilo de ejecución.

Por último, en el caso de que se haya realizado la conexión de forma correcta, iniciaremos el hilo de ejecución referente a la gestión de la conexión.

d) ConnectedThread.java.

El hilo de ejecución ConnectedThread se encarga de la gestión de la conexión, es decir, maneja el BluetoothSocket creado anteriormente. Destacamos entre sus atributos:

```
private final BluetoothSocket mmSocket;
private final InputStream mmInStream;                              (1)
private final OutputStream mmOutStream;                            (2)
```

1) *InputStream*: gestiona el *buffer* de datos de entrada.
2) *OutputStream*: gestiona los datos de salida que queramos enviar.

Dentro de esta clase, debemos diferenciar dos métodos destacables:

- En el hilo de ejecución ***run()*** se encuentra el método ***read(bytes[] buffer)*** que lee los bytes del *buffer* de entrada. Esto sirve para tener un hilo de ejecución que constantemente lea los datos que nos vayan llegando al receptor.

- Si queremos enviar datos solamente tenemos que llamar a uno de los métodos ***write(bytes[]* byte*)*** o bien ***write(String* message*).***

e) Llamada al servicio desde una actividad.

Para ello, debemos hacer uso de la instancia de IBinder que hemos creado antes dentro de la actividad del efecto que queramos.

Creamos una instancia de ServiceConnection que nos permite enlazar la actividad al servicio:

```
public class BlinkActivity extends AppCompatActivity{

//...

private ServiceConnection mConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName className,
                IBinder service) {
        BluetoothService.LocalBinder binder =
                    (BluetoothService.LocalBinder) service;
        btService = binder.getService();
        mBound = true;
    }

//… Resto de la actividad
}
```

Una vez tengamos enlazada actividad y servicio, podemos llamar a los métodos del propio servicio mediante el Binder.

Dentro de la declaración del botón de Enviar, tendríamos para escribir:

```
if (mBound){


btService.conexion.mConnectedThread.write(…);
      }
```

Con el método ***write(String s***), podemos enviar ya la información que deseemos por Bluetooth.

Una vez realizado esto, la aplicación podrá enviar vía Bluetooth la trama completa para su recepción en el sistema Raspberry Pi.

6. Pruebas y validaciones realizadas

A lo largo del proyecto se han realizado diversas pruebas para comprobar la funcionalidad de cada una de las partes y del sistema completo.

6.1. Depuración del programa principal.

Para programar en Raspberry Pi se ha utilizado el sencillo entorno de desarrollo gráfico Geany. Como la funcionalidad de Geany es limitada, a la hora de depurar el código se han incluido trazas para comprobar la funcionalidad del mismo.

Las trazas actúan como logs para mostrar por consola el correcto funcionamiento del código. Se implementan utilizando la función ***printf().***

6.2. Puerto serie y Bluetooth.

En cuanto a la funcionalidad del puerto serie, se ha escrito un programa de test para comprobar su funcionalidad, tanto en escritura como lectura.

Utilizando la librería wiringSerial y un sencillo código en C++, se comprueban los bytes en el *buffer* del puerto serie y se comparan con los enviados.

Desde la aplicación Android se comparan utilizando la aplicación Bluetooth SPP pro, que permite el envio y recepción de datos Bluetooth con el perfil del puerto serie.

Para ello se ha creado una aplicación de test, *bluetooth.cpp*, a ejecutar en la Raspberry Pi. Dicha aplicación obtiene todos los caracteres que recibe en el pin RXD, los presenta en pantalla, los almacena en un array y cuando recibe un carácter de terminación 'A', presenta todos los recibidos.

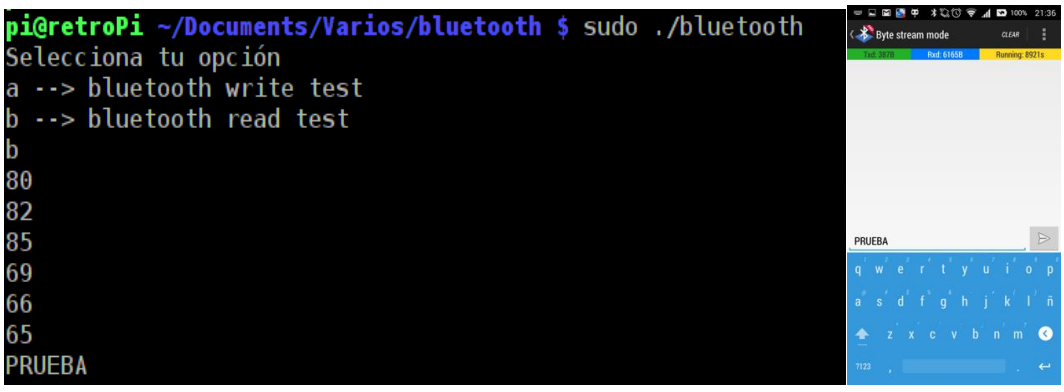

Ilustración 10 y 11. bluetooth.cpp

6.3. Pruebas hardware.

Para testear el hardware del proyecto se han comprobado las señales mediante un multímetro (para señales continuas) y un osciloscopio.

La placa Bulletin LED Board ya se encontraba previamente testeada, por lo que sencillamente se ha comprobado que

Se comprueba el funcionamento correcto de las señales GSclk, Sin y Sclk mediante una sonda. Todas estas señales son digitales y debemos ver los pulsos correctamente.

Los pasos realizados para llevar a cabo las pruebas con la placa son los siguientes:

- Soldadura de todos los componentes de la misma, además de los zócalos en los cuales se ubicarán los pines de la Raspberry Pi y los pines tanto de salida de los drivers como de alimentación, puerto serie (UART) de la Raspberry.

- Crimpado y colocación de los conectores de las tiras de LED.

- Pruebas y validaciones de la placa, ya mencionadas anteriormente.

7. Conclusiones y futuras líneas de trabajo.

Como conclusión a la memoria anteriormente expuesta y en comparación a los requisitos establecidos previamente a la realización del proyecto, podemos dar por cumplidos los objetivos de:

- Desarrollo y codificación de diversos efectos básicos. Creadas las bases para una codificación sistemática de efectos en el futuro, mediante la documentación referente a la función update(), así como las funciones básicas creadas y la implementación del modelo de datos que se utiliza en la comunicación entre la Raspberry y el dispositivo Android.

- Diseño e implementación de la recepción de datos por Bluetooth en la Raspberry Pi con el módulo HC-05. Creación de los hilos de ejecución en el programa principal en Raspberry, uno de ellos dedicado a la lectura del puerto serie. Diseño e implementación del servicio Bluetooth en Android. Diseño del modelo de datos basado en *Strings* para realizar la comunicación entre ambos dispositivos.

- Diseño y desarrollo de una aplicación Android sencilla para la interactividad tablón – usuario. Diseño e implementación de la interfaz de usuario de la aplicación. Diseño e implementación de un servicio Android que permita la gestión en segundo plano de la conectividad Bluetooth.

Una vez cumplidos estos objetivos, se ha intentado en medida de lo posible dotar de una estética agradable a la interfaz de usuario, además de sistematizar la programación de nuevos efectos de luces y de envío de datos desde la *app* móvil.

Desde un punto de vista personal, sería interesante poder continuar desarrollando la aplicación y las posibilidades del tablón, una vez han sido sentadas las bases para poder crear nuevas funcionalidades de manera sencilla y práctica.

Se ha realizado el proyecto con una idea esencial, que es la posible ampliación de estas funcionalidades en el futuro. Se han sentado las bases para ampliar, sobre todo a nivel de software, tanto en las fases referidas a la Raspberry como en la aplicación Android, las posibilidades que la iluminación del tablón ofrece.

- Futuras líneas de trabajo.

La integración entre la aplicación de Android y la Raspberry Pi resulta sencilla gracias a la trama específica creada a tal efecto. Los parámetros de configuración de efectos son convertidos automáticamente a su color, duración, tipo de efecto, etc, según son recibidos en el sistema empotrado.

Entre otros, se propone la continuación de los siguientes puntos:

- Creación de nuevos efectos de luces. Partiendo de los efectos básicos *on_segment* y *on_segment_timed* resulta sencillo crear nuevos efectos de luces siempre que sean de iluminación constante. Para crear efectos con degradados o niveles de luz diferente puede partirse de *kitt_effect* o *dimmerUp.*

- Mejora de la interfaz de usuario. Posibilidad de incluir interacción gráfica con una imagen del tablón, pudiendo elegir de forma intuitiva la iluminación del mismo.

- Envío de datos por Bluetooth desde la Raspberry a la app de Android, pudiendo mostrar información sobre el estado en el que se encuentran las luces. Ampliación del mismo añadiendo más funcionalidades, utilizando por ejemplo el sensor de presencia y calculando estadísticas de tránsito de gente por el pasillo o de uso del tablón y mostrándolas en la app.

- El nuevo modelo de Raspberry Pi 3 incorpora un módulo Bluetooth propio en su chipset. De cara a integrar en una única Raspberry Pi todas las funcionalidades del tablón, tanto la aplicación Android como el sistema de pulsadores que posee (el cual hace uso de la UART física de la misma), se integran las funcionalidades Bluetooth en el nuevo modelo.

  Para realizar esto la migración es sencilla. Una vez actualizados los drivers respectivos en la nueva Raspberry, el Bluetooth utiliza el mismo puerto serie y con la misma denominación que el que hemos usado en la Raspberry Pi 2, por lo que la migración es inmediata.

  En la aplicación Android debemos cambiar la MAC a utilizar a la hora de realizar la conexión por la nueva.

- Creación de una base de datos de usuarios, en el que cada integrante del laboratorio pudiera personalizar a su gusto los efectos preferidos que desee tener al entrar en el mismo, utilizando sus credenciales del mismo.

- Inclusión de diversos juegos o funcionalidades interactivas parte de otro Trabajo Fin de Grado realizado con pulsadores. Integración de los mismos en la aplicación Android.

- Posibilidad de crear un tablón virtual en la aplicación que muestre igualmente las novedades y proyectos del laboratorio.

- Ampliación de la aplicación a una aplicación general B105 Electronic Systems Lab con conectividad al futbolín (web de estadísticas, etc), la aplicación de la red de sensores incluida en otro trabajo de Fin de Grado y el servidor web del laboratorio (Wiki, documentos…), así como las funcionalidades del tablón y el posible tablón virtual.

- Integración con una matriz de LED que permita desplegar mensajes en el tablón y pudiendo personalizarse los mismos desde la aplicación.

- Optimización del rendimiento de la conectividad Bluetooth, buscando otras alternativas a los dos hilos (*polling*) utilizando interrupciones u otros métodos.

- Optimización del rendimiento de la conectividad Bluetooth, buscando otras alternativas a los dos hilos (*polling*) utilizando interrupciones u otros métodos.

Tabla de acrónimos.

Acrónimo	Significado
API	Application Programming Interface
GPIO	General Purpose Input/Output
GS	Greyscale
LED	Light Emitting Diode
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
RGB	Red, Green, Blue
ROM	Read Only Memory
SMD	Surface Mount Device
SPP	Serial Port Protocol
UART	Universal Asynchronous Receiver-Transmitter
VNC	Virtual Network Computing

Bibliografía y referencias.

[1] Documentación de la librería TLC5940 para Raspberry Pi, de la página web *return1*. http://return1.net/projects/tlc5940-raspberry-pi-library/

[2] B-105. Placa Bulletin LED Board. Esquemático.

[3] Texas Instruments. Datasheet. TLC5940. http://www.ti.com/lit/ds/symlink/tlc5940.pdf

[4] Web con información sobre el código de colores en hexadecimal. http://www.color-hex.com/

[5] Información sobre el módulo HC-05. Wiki de la página *iteadstudio*. http://wiki.iteadstudio.com/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05#Overview

[6] Documentación de la librería *WiringPi* para la gestión de los GPIO en Raspberry. http://wiringpi.com/

[7] Documentación de la cabecera *fcntl.h* http://pubs.opengroup.org/onlinepubs/009695399/basedefs/fcntl.h.html

[8] Detalle de la cuota de mercado de Android. http://www.lavanguardia.com/tecnologia/20160527/402070812723/android-crece-cuota-de-mercado.html

[9] Referencia a la API de Bluetooth de Google. Guía de desarrolladores Android. https://developer.android.com/guide/topics/connectivity/bluetooth.html?hl=es

[10] Jesús Tomás Gironés, *El gran libro de Android* – 3ª edición,. Ed. Marcombo.

[11] Eben Upton, Gareth Halfacree, *Raspberry Pi User Guide* – 2nd edition.

[12] Brendan Horan, *Practical Raspberry Pi* – 2013 edition.

[13] Curso de programación en C++, http://c.conclase.net/curso/?cap=003#inicio